

SAS Programming Fundamentals II

**DCS/TASC/Advanced Support Team
Center for Information Technology
National Institutes of Health**

Summer 2001

SAS Programming Fundamentals II

This course builds on skills taught in (212) - SAS Programming Fundamentals I. Its focus is on expanding Base SAS programming skills by including the DATA step management components of the Base SAS software...how to manipulate SAS data set effectively. Once you have completed both courses(212) and (213) you have a strong programming foundation for becoming a successful Base SAS programmer.

Many software applications are either totally menu driven, or totally command driven ("enter a command-see the result"). Base SAS software is neither totally menu driven or totally command driven. **With Base SAS software, you use statements to write a series of instructions called a SAS program.**

MODULE 4: DATA STEP STATEMENTS

- Assignment Statement
- Identify Arithmetic Operators
- Interpret Messages in the SAS Log
- SAS Functions—SUM, MEAN, SUBSTR, TRIM, MDY, MONTH, DAY, YEAR
- RETAIN and SUM Statements

MODULE 5: CONDITIONAL STATEMENTS

- IF-THEN Statement
- Comparison, Logical and IN Operators
- IF-THEN/ELSE statements
- Subsetting IF Statement
- IF-THEN/DELETE Statement
- OUTPUT Statement
- LENGTH Statement

MODULE 6: MANAGING SAS DATA SETS

- PROC SORT
- SET statement
- MERGE statement
- WHERE statement
- DROP = and KEEP = SAS Data Set Options

MODULE 4: DATA STEP STATEMENTS

- Assignment Statement
- Identify Arithmetic Operators
- Interpret Messages in the SAS Log
- SAS Functions—SUM, MEAN, SUBSTR, TRIM, MDY, MONTH, DAY, YEAR
- RETAIN and SUM Statements

1. WRITE ASSIGNMENT STATEMENTS

Purpose: Use assignment statements in the DATA step to create new variables and to modify existing variables. Assignment statements evaluate an expression and store the result as a variable.

General Form: variable = expression ;
 | |
 (A) (B)

(A) variable names a new or existing variable

(B) expression may consist of one or more variable names, constants, function names, and arithmetic operators

An assignment statement is one of the few statements in SAS that does not start with a keyword.

Sample SAS statements:

Notice that these samples contain numeric, character, and date constants.

(1) WTKILO = WEIGHT * .45 ;

(2) ID = SUBSTR(ID, 2) ;

(3) TOTAL_COST = (PPRICE * PNUM) + (CPRICE * CNUM) ;

(4) REMARKS = 'OK' ;

(5) SUM = SUM + 1 ;

(6) Date_of_birth = '04MAY67'D ; /*a SAS date constant */

Sample: Using an Assignment Statement

```
DATA INCOME ;  
  INPUT  RENT_INCOME  EXPENSES ;  
  TOTAL_INCOME = RENT_INCOME - EXPENSES ;  
  
  /*other SAS programming statements*/  
  
DATALINES ;  
6350 1200  
7950 1300  
;  
  
PROC PRINT;  
  TITLE 'Using An Assignment Statement' ;  
RUN;
```

Sample Output:

Using An Assignment Statement			
OBS	RENT_INCOME	EXPENSES	TOTAL_INCOME
1	6350	1200	5150
2	7950	1300	6650

What happens during SAS's "execute" phase of this particular assignment statement

TOTAL_INCOME = RENT_INCOME - EXPENSES ;

STEP 1: Each variable is initialized to missing at the start of the "execute" phase

PDV

N	_ERROR_	RENT_INCOME	EXPENSES	TOTAL_INCOME
		.	.	.

STEP 2: INPUT statement reads the first record.

PDV

N	_ERROR_	RENT_INCOME	EXPENSES	TOTAL_INCOME
1	0	6350	1200	.

STEP 3: SAS evaluates TOTAL_INCOME = RENT_INCOME - EXPENSES.

PDV

N	_ERROR_	RENT_INCOME	EXPENSES	TOTAL_INCOME
1	0	6350	1200	5150

STEP 4: By default, the information in the PDV is written as an observation to the SAS data set INCOME at the end of the DATA step. The same cycle is repeated for the next record.

2. IDENTIFY ARITHMETIC OPERATORS

Arithmetic Symbol	Operation	Sample ASSIGNMENT statements
+	addition	SUM = X + Y
—	subtraction	DIFFERENCE = X - Y ;
*	multiplication	PRODUCT = X * 6 ;
/	division	RATE = DISTANCE/TIME ;
**	exponentiation	AREA = SIDE ** 2 ;

EXERCISE 1

Which of the following is an invalid SAS assignment statement? (circle)

- (1) $X = A * B ;$
- (2) $Z = 15/0 ;$ (a SAS program using division by 0 will be discussed as we move along)

Note: *SAS does compile and execute the SAS statement. However, the SAS Log note reads..."NOTE: Division by Zero detected at line (number) and column (number).*

a second SAS Log note reads...

NOTE: *Mathematical operations could not be performed in the following place: The results of the operations have been set to missing values.*

- (3) $3 * (A - B) ;$
- (4) $DAY_WAGE = (HOUR/8) * WAGERATE;$
- (5) $X \text{ SQUARE} = X ** 2;$
- (6) $TOTAL = TOTAL + 1;$
- (7) $PERCENT = PERCENT * 100;$
- (8) $RANGE = HIGHTEMP - LOWTEMP;$
- (9) $D = A/B * 100$

2.1 Missing values in an assignment statement

Should the value for a variable in an expression be missing, the resulting value will also be missing.

Sample Program:

```
DATA MISSING ;
    INPUT A B ;
    TOTAL = A + B ;
DATALINES;
2 .
3 5
;

PROC PRINT ;
    TITLE 'Effect of missing values on an ASSIGNMENT statement' ;
RUN;
```

Sample SAS Log:

```
1      DATA MISSING ;
2          INPUT A B ;
3          TOTAL = A + B ;
4      DATALINES;
```

NOTE: Missing values were generated as a result of performing an operation on missing values.

Each place is given by: (Number of times) at (Line):
(Column)1 at 3:13

NOTE: The data set WORK.MISSING has 2 observations and 3 variables.

NOTE: The DATA statement used
cpu time 0.06 CPU seconds.

```
7      ;
```

```
8      PROC PRINT ;
```

```
9          TITLE 'Effect missing values on an ASSIGNMENT statement';
```

NOTE: The PROCEDURE PRINT printed page 1.

NOTE: The PROCEDURE PRINT used:
cpu time 0.02 CPU seconds.

Sample Output:

Effect missing values on an ASSIGNMENT statement

OBS	A	B	TOTAL
1	2	.	.
2	3	5	8

3. INTERPRET MESSAGES IN THE SAS LOG

3.1 Variable is Declared Character and Used as Numeric

Sample Program:

```
DATA DEFCHAR ;  
    INPUT CODE $3. ;  
    CODE = CODE + 1 ;  
DATALINES;  
624  
239  
;  
  
PROC PRINT ;  
RUN;
```

Sample SAS Log:

```
1          DATA DEFCHAR ;  
2              INPUT CODE $3. ;  
3              CODE = CODE + 1 ;  
4          DATALINES;
```

NOTE: Character values have been converted to numeric values at the places given by:
(Number of times) at (Line):(Column).
2 at 3:11

NOTE: Numeric values have been converted to character values at the places given by:(Number of times)
at (Line):(Column). 2 at 3:16

NOTE: The data set WORK.DEFCHAR has 2 observations and 1 variables.

NOTE: The DATA statement used:
cpu time 0.06 CPU seconds.

```
7          ;  
8          PROC PRINT ;
```

NOTE: The PROCEDURE PRINT printed page 1.

NOTE: The PROCEDURE PRINT used:
cpu time 0.02 CPU seconds.

Sample Output:

The SAS System

OBS	CODE
1	625
2	240

3.2 Division by Zero

Sample Program:

```
DATA DIVZERO ;
  INPUT X Y ;
  Z = X/Y ;
DATALINES ;
4 1
6 0
;
PROC PRINT;
RUN;
```

Sample SAS Log:

```
1          DATA DIVZERO ;
2          INPUT X Y ;
3          Z = X/Y ;
4          DATALINES;
```

ERROR: Division by zero detected at line 3 column 9.

```
RULE:      ----+-----1-----+-----2-----+-----3-----+-----4-----+-----5---
6          6 0
```

X=6 Y=0 Z=. _ERROR_=1 _N_=2

NOTE: Mathematical operations could not be performed at the following places. The results of the operations have been set to missing values.

Each place is given by:

(Number of times) at (Line):(Column).

1 at 3:9

NOTE: The data set WORK.DIVZERO has 2 observations and 3 variables.

NOTE: The DATA statement used 0.06 CPU seconds.

```
7          ;
8          PROC PRINT ;
```

NOTE: The PROCEDURE PRINT printed page 1.

NOTE: The PROCEDURE PRINT used
cpu time 0.02 CPU seconds.

ERROR: Errors printed on page 1.

Sample Output:

OBS	X	Y	Z
1	4	1	4
2	6	0	.

EXERCISE 2

```
(1) DATA REPORT;
    INPUT GRANT TOTAL1 TOTAL2;
    TOTAL = TOTAL1 + TOTAL2;
    /*later we will discuss usage of both the RETAIN and Assignment statements*/
    DATALINES;
    65 40 53
    86 70 .
    ;
    PROC PRINT;
    RUN;
```

Sample SAS Log:

```
1          DATA REPORT;
2          INPUT GRANT TOTAL1 TOTAL2;
3          TOTAL = TOTAL1 + TOTAL2;
4          DATALINES;
NOTE: Missing values were generated as a result of performing
      an operation on missing values.
      Each place is given by:
      (Number of times) at (Line):(Column).
      1 at 3:18
NOTE: The data set WORK.REPORT has 2 observations and 4
      variables.
NOTE: The DATA statement used
      cpu time          0.06 seconds.

7          ;
8          PROC PRINT;

NOTE: The PROCEDURE PRINT printed page 1.
NOTE: The PROCEDURE PRINT used 0.02 CPU seconds.
NOTE: The SAS session used:
      cpu time          0.24 seconds.
```

Examine the following SAS output. Why is SAS generating missing values?

The SAS System				
OBS	GRANT	TOTAL1	TOTAL2	TOTAL
1	65	40	53	93
2	86	70	.	.

```

(2) DATA AVERAGE;
      INPUT TIME1 TIME2 TIME3 N ;
      AVG = (TIME1 + TIME2 + TIME3)/N;
      DATALINES;
      14.5 18 23.3 3
      13.0 13.2 15 3
      18.4 21 25.5 0
      ;

      PROC PRINT;
      RUN;

```

What is causing an error message in the following SAS Log ?

```

1          DATA AVERAGE;
2          INPUT TIME1 TIME2 TIME3 N ;
3          AVG=(TIME1+TIME2+TIME3)/N ;
4          DATALINES;

ERROR: Division by zero detected at line 3 column 26.
RULE:      ----+----1----+----2----+----3----+----4----+----5----
7          18.4 21 25.5 0
TIME1=18.4 TIME2=21 TIME3=25.5 N=0 AVG=. _ERROR_=1 _N_=3
NOTE: Mathematical operations could not be performed at the
      following places. The results of the operations have been
      set to missing values.
      Each place is given by:
      (Number of times) at (Line):(Column).
      1 at 3:26
NOTE: The data set WORK.AVERAGE has 3 observations and 5
      variables.
NOTE: The DATA statement used:
      cpu time          0.06 seconds.

8          ;
9          PROC PRINT;
NOTE: The PROCEDURE PRINT printed page 1.
NOTE: The PROCEDURE PRINT used:
      cpu time          0.03 seconds.
ERROR: Errors printed on page 1.

```

The SAS System

OBS	TIME1	TIME2	TIME3	N	AVG
1	14.5	18.0	23.3	3	18.6
2	13.0	13.2	15.0	3	13.7
3	18.4	21.0	25.5	0	.

4. SAS Functions

Purpose: The SAS System contains functions which perform specific calculations. Functions operate across variables within the observations in a SAS data set.

General Form: variable = functionname (argument1, ..., argumentn) ;
 | | |
 (A) (B) (C)

(A) variable result of the function

(B) functionname keyword for a particular function

(C) argumentn any SAS expression

4.1 SUM Function

Purpose: The SUM function adds its non-missing arguments.

General Form: SUM(argument1, . . . , argumentn) ;

Sample: To illustrate the difference between the usage of the SUM function versus the addition operator in the ASSIGNMENT statement when missing values are present.

```
DATA Sample ;
  INPUT X1 X2 X3 ;

  TOTAL1 = SUM(X1, X2,X3) ;

  /* the expression SUM(X1, X2, X3) is equivalent to the expression SUM(OF X1-X3) */

  /* variables which start with a common prefix and end with consecutive numbers can be
  part of a numbered range list. The numbers can start and end anywhere as long as the
  number sequence between is complete. */

  TOTAL2 = X1 + X2 + X3 ;

DATALINES;
1 5 10
2 . 4
;

PROC PRINT ;
  Title 'Notice the difference between the values TOTAL1 and TOTAL2' ;
RUN;
```

Sample 2 - SAS Output:

Notice the difference between the values TOTAL1 and TOTAL2

OBS	X1	X2	X3	TOTAL1	TOTAL2
1	1	5	10	16	16
2	2	.	4	6	.

4.2 INPUT Function - Explicit Character-to-Numeric Conversion

The INPUT function converts character data values to numeric values.

General Form: INPUT (***source***, informat) ;

- ***source*** indicates the character variable, constant, or expression to be converted to a numeric value
- a numeric ***informat*** must be specified

Suppose: The variable **PayRate** is stored in a SAS data set as a character variable. You are asked to determine the salary. Multiple variable **PayRate** by the variable **Hours**.

Given: PayRateHours

10	88
8	200

Sample Program:

```
Data Convert_Char_to_Numeric ;
  INPUT  PayRate $ Hours ;
  New_PayRate = INPUT(PayRate, 2.) ;
  Salary = New_PayRate * Hours ;
DATALINES;
10      88
8       200
;

PROC PRINT;
RUN;
```

/*Alternatively, you may combine into one Assignment statement to accomplish the task*/

```
Data Convert_Char_to_Numeric ;
  INPUT  PayRate $ Hours ;
  Salary = INPUT(PayRate, 2.) * Hours ;
DATALINES;
10      88
8       200
;
```

4.3 PUT Function - Explicit Numeric-to-Character Conversion

The PUT function converts numeric data values to character values.

General Form: PUT(***source***, format) ;

- ***source*** indicates the numeric variable, constant, or expression to be converted to a character value
- a **format** matching the data type of the source

Suppose: You were asked to create a new character variable **Assignment**. This new variable would be the result of concatenating the values of the variables **Site** and **Department**.

Given:

Department	Site
PURH	57
BK	34

Sample Program:

```
Data Convert_Numeric_to_Character ;
```

```
INPUT Department $ Site ;
```

```
Newsite =PUT(Site, 2.) ;
```

```
/*this statments converts the numeric variable Site to a character variable Newsite */
```

```
Assignment = Newsite || '/' || Department ;
```

```
DATALINES;  
PURH 57  
BK 34  
;
```

```
PROC PRINT;  
RUN;
```

Sample Output:

Obs	Department	Site	Newsite	Assignment
1	PURH	57	57	57/PURH
2	BK	34	34	34/BK

/*Alternatively, you may combine into one Assignment statement to accomplish the task*/

```
Data Convert_Numeric_to_Character ;
```

```
    INPUT  Department    $    Site  ;
```

```
    Assignment = PUT (Site, 2.) || '/' || Department ;
```

```
DATALINES;
```

```
PURH      57
```

```
BK        34
```

```
;
```

```
PROC PRINT;
```

```
RUN;
```

Sample Output 2:

Obs	Department	Site	Assignment
1	PURH	57	57/PURH
2	BK	34	34/BK

4.4 MEAN Function

Purpose: The MEAN function averages its **non-missing** arguments.

General Form: MEAN(argument1,.. .,argumentn) ;

You may use the **numbered range lists** in an abbreviated form... MEAN(OF varlist) ;

Sample 1: M = MEAN(2, 6, 7) ;

Sample 2: Illustrating the difference between usage of the MEAN function in a SAS expression and the usage of addition/division operators in a SAS expression when missing values occur.

```
DATA AVG ;
```

```
INPUT X1 X2 X3 X4 ;
```

```
AVG1 = MEAN(OF X1-X4) ; /* or AVG1 = MEAN(X1, X2, X3, X4); */
```

```
/* The expression AVG1 = MEAN(OF X1-X4) is equivalent to AVG1= MEAN(X1, X2, X3, X4) */
```

```
AVG2 = (X1 + X2 + X3 + X4)/4 ;
```

```
DATALINES;
```

```
2 6 7 .
```

```
;
```

```
PROC PRINT ;
```

```
Title 'Notice the difference between the values AVG1 and AVG2' ;
```

```
Title2 'Usage of the MEAN function in a SAS expression versus usage of the' ;
```

```
Title3 '+ or - operators in a SAS expression when missing values occur.' ;
```

```
RUN;
```

Sample 2 - SAS Output:

Notice the difference between the values AVG1 and AVG2
Usage of the MEAN function in a SAS expression versus usage of the
+ or - operators in a SAS expression when missing values occur.

OBS	X1	X2	X3	X4	AVG1	AVG2
1	2	6	7	.	5	.

4.5 TRIM Function is a character-handling function

Purpose: The TRIM function removes trailing blanks from a character string. TRIM function is handy for making concatenated strings look more presentable.

General Form: **TRIM**(string) ;

The TRIM function is often used in conjunction with the concatenation operator (||). The concatenation operator joins character strings and has the general form:

string1 || string2

where **string1** and **string2** are character variables, constants, or expressions.

Note 1:

The concatenation operator (||) enables you to append character variables and/or character constants to each other.

Trailing blanks are not trimmed from the operands.

For instance, the value “East” stored in a variable with a length of 10 will contribute six blanks to the concatenated string. The TRIM function eliminates this unwanted “white space.”

Note 2:

On the keyboard the || symbol is the uppercase of the back-slash.

Sample Program: TRIM function

```
DATA NAMES ;  
  
  INPUT FNAME $ LNAME $ ;  
  
  /* Used several Assignment statements below: */  
  
  _PART1 = FNAME || LNAME ;  
  
  _PART2 = TRIM(FNAME) || LNAME ;  
  
  FULL_NAME = TRIM(FNAME) || ' ' || LNAME ;  
  
DATALINES;  
HARRY    THOMAS  
;  
  
PROC PRINT ;  
RUN;
```

Sample Output:

OBS	FNAME	LNAME	_PART1	_PART2	FULL_NAME
1	HARRY	THOMAS	HARRY THOMAS	HARRYTHOMAS	HARRY THOMAS

/* Alternatively, you may combine into one Assignment statement to accomplish the task */

DATA NAMES ;

INPUT FNAME \$ LNAME \$;

FULL_NAME = TRIM(FNAME || ' ' || LNAME) ;

DATALINES;

HARRY THOMAS

;

PROC PRINT ;

RUN;

Sample Output:

OBS	FNAME	LNAME	FULL_NAME
1	HARRY	THOMAS	HARRY THOMAS

4.6 SUBSTR Function is a character-handling function

Purpose The SUBSTR function extracts part or all of a character string.

General Form: **SUBSTR**(string, start, length) ;
 | | |
 (A) (B) (C)

- | | |
|------------|---|
| (A) string | string and character string are used interchangeably and refer to any of character variables, constants, or expressions. |
| (B) start | extraction begins in the specified 'start' position from the left and continues to the right for the specified ' <i>length</i> ' |
| (C) length | the number of characters to extract. If length is omitted or exceeds string's length, the remainder of the character string is extracted. |

Sample program 1 - SUBSTR function:

```
DATA Substring_Program_1 ;
```

```
    INPUT  FirstName $ 1-9    @13 Middle_Name $9. ;
```

```
    Middle_Initial = SUBSTR(Middle_Name,1,1) ;
```

```
    /* in this example extraction of a string starts with the 1st character in the field and continues for  
       1 position*/
```

```
DATALINES;
```

```
Elizabeth    Marie  
Richard      Lee  
Brian        Thomas  
;
```

```
PROC PRINT ;
```

```
RUN;
```

Sample output:

Obs	First Name	Middle_ Name	Middle_ Initial
1	Elizabeth	Marie	M
2	Richard	Lee	L
3	Brian	Thomas	T

Sample program 2 - SUBSTR function:

```
DATA Substring_Program_2 ;  
    INPUT  Variable_1  Unknown_ID  $5. ;  
    ID_variable = SUBSTR(Unknown_ID,3,3) ;  
    /*extracting a character string begins at the 3rd character in the original string  
    and from that point extracts the following 3 positions */  
  
DATALINES;  
10      SA052  
20      MA011  
;  
  
PROC PRINT ;  
RUN;
```

Sample output:

OBS	Variable_1	Unknown_ID	ID_variable
1	10	SA052	052
2	20	MA011	011

4.7 Date Functions

Sometimes dates are supplied in a form not amenable to any of the SAS date informats. (You studied SAS date informats in the course SAS Programming Fundamentals I. Certain situations require date calculations. These date creation activities are often handled by the SAS **date functions**.

4.5.1 MDY Function

Purpose: The MDY function creates a SAS date from separate month, day, and year arguments.

General Form: MDY(month,day,year) ;

SAS numeric variables or constants represent month, day, and year, respectively. A missing or out-of-range argument creates a missing value.

Sample:

```
DATA TEST ;  
  INPUT ID 1-5  MNTH 7-8  DY 10-11  YR 1-2 ;  
  NEWDATE = MDY(MNTH,DY,YR) ;  
DATALINES;  
91215 12 24  
;
```

```
PROC PRINT ;  
  /* Without a FORMAT statement, SAS writes 11680. This integer 11680 represents  
    the total number of days between Dec. 24, 1991 and Jan. 1, 1960 */  
RUN;
```

Sample Output:

OBS	ID	MNTH	DY	YR	NEWDATE
1	91215	12	24	91	11680

4.7.2 MONTH, DAY, YEAR Functions

Using SAS functions to extract month, day, and year from a SAS date value.

FUNCTION	PURPOSE	GENERAL FORM
MONTH	Returns the numeric value of the month (1-12)	MONTH(SASdate)
DAY	Returns the day of the month	DAY(SASdate)
YEAR	Returns the year in 4 digits	YEAR(SASdate)

Sample - YEAR, MONTH, and DAY functions program

```
DATA DOB ;  
  DOB = '07JAN1964'D ; /* this is a SAS date constant */  
  YR = YEAR(DOB) ;  
  MON = MONTH(DOB) ;  
  DAY = DAY(DOB) ;  
  
PROC PRINT ;  
RUN;
```

Sample Output:

OBS	DOB	YR	MON	DAY
1	1467	1964	1	7

5. RETAIN, ASSIGNMENT and SUM STATEMENTS

5.1 RETAIN Statement

Purpose: The RETAIN statement causes a variable whose value is assigned by an INPUT or ASSIGNMENT statement to retain its value from one iteration of the DATA step to the next. Without a RETAIN statement, the SAS System automatically sets these variables to missing before each iteration of the DATA step.

General Form: RETAIN variable [initial-value];

|
(A)

(A) initial-value variable(s) listed before an initial-value will start the first iteration of the Data step with that value

The RETAIN statement can appear anywhere in the DATA step.

The RETAIN statement may be desirable in some situations. For instance, you may want to compute a running total for one or more variables. But by default SAS automatically resets all values to missing at each iteration of the DATA step. In such a case you want the value for the running total to be retained at the start of each iteration.

Basically...

1. To provide initial value of your choice for the variable(s)
2. To prevents resetting value to missing for each iteration of the Data Step

Sample 1: Using the RETAIN statement and the ASSIGNMENT statement

```
DATA TOTCOST ;  
  RETAIN TOTAL 0 ;    /* TOTAL is initialized to 0 */  
  INPUT COST ;  
  TOTAL = TOTAL + COST ; /* TOTAL keeps a 'running sum' */  
DATALINES;  
1000  
1200  
;
```

PDV Before Reading the First Data Line

TOTAL	COST
0	

PDV After Reading the First Data Line

TOTAL	COST
1000	1000

PDV Before Reading the 2nd Data Line

TOTAL	COST
1000	

PDV After Reading the 2nd Data Line

TOTAL	COST
2200	1200

Before the first execution of the DATA step, the variable TOTAL is initialized to 0. Then for all subsequent executions, the RETAIN statement signals to SAS that the variable TOTAL is not initialized to missing. Therefore, you can accumulate totals using the RETAIN statement and the assignment statement as long as the variable COST is not missing. Should COST be a missing value then the value for TOTAL will also be missing.

Sample 2: Using the RETAIN statement and the ASSIGNMENT statement

```
DATA DAYS;  
  
    RETAIN TODAY '28FEB92'D;    /* the internal value for TODAY is retained */  
                                /* '28FEB92'D is a date constant */  
  
    INPUT DATE MMDDYY6.;        /* for each iteration of the DATA step */  
  
    Number_Days = TODAY - DATE;  
  
DATALINES;  
010192  
021586  
;  
  
PROC PRINT;  
RUN;
```

Sample Output:

OBS	TODAY	DATE	Number_Days
1	11746	11688	58
2	11746	9542	2204

5.2 SUM Statement

Purpose: The **Sum** statement is a special SAS statement which also retains the variable's value from the previous iteration of the DATA step. SAS recognizes the first variable in the SUM statement as **a retained numeric variable**. Retention is implied. But you use it for the special cases where you simply want to cumulatively add the value of an expression to a variable.

General Form: variable + expression ;
 | |
 (A) (B)

(A) variable This variable keeps the 'running' sum. By default this numeric variable is given an initial value of zero.

(B) expression The expression is evaluated and the result is added to the value of the variable.

Sample program:

```
DATA ADDIT ;  
  INPUT COST ;  
  TOTAL + COST ;
```

/* by definition TOTAL represents the variable and COST represents the expression*/

```
DATALINES;  
1000  
1200  
.  
1500  
;
```

```
PROC PRINT ;  
RUN;
```

Sample output:

OBS	COST	TOTAL
1	1000	1000
2	1200	2200
3	.	2200
4	1500	3700

EXERCISE 3 (computer-assisted)

1. Write a DATA step that reads the following data lines.

BOB	34	31	22
JAN	23	22	18
MATT	12	43	25
MARY	15	51	40

- (1a) Use the following INPUT statement in the DATA step.
INPUT NAME \$ GR1 GR2 GR3 ;
 - (1b) Create a new variable TOTGR which is the sum of GR1, GR2 times 5, and GR3.
 - (1c) Create another variable AVGGR which is the average of GR1, GR2 times 5, and GR3.
2. Use PROC PRINT to list the observations and variables created in the DATA step(above).
 3. Write another DATA step that reads company names and phone numbers.

DATA PHONES ;

INPUT COMPANY \$ 1-5 PHONE_NUMBER \$ 8-14;

/*your SAS statements and answers to questions (3a) and (3b) will be entered here, that is, immediately after the INPUT statement and before the DATALINES statement*/

DATALINES ;

C & P 4651500

PEPCO 5401800

WSSC 8451000

;

- (3a) Using the SUBSTR function, write ASSIGNMENT statements, and create two new variables.

The variable **EXCHANGE** will represent the first three digits of PHONE_NUMBER and the variable **EXTENSION** will represent the last four digits of PHONE_NUMBER

NOTE! *You determine the values for the 'start' and the 'length' values in the SUBSTR function by simply counting the number of digits or characters in a specified field.*

- (3b) Create a new variable **NEWPHONE_NUMBER** (...insert a hyphen between the three digit exchange and the four digit extension).
- (4) Use PROC PRINT to display the observations and variables in the data set **PHONES**
- (5) Check the SAS log and the SAS output. Is this the desired SAS output ?

EXERCISE 4 (computer-assisted)

1. Write a DATA step that reads the following data lines. Each data line consists of a patient's identification number, followed by the month, day and year that these patients were admitted to a hospital.

```
1069 12 10 81
1200 5 8 82
1180 6 14 82
```

- (a) Use the following INPUT statement in the DATA step

```
INPUT PATID $  ADMISSION_MONTH  ADMISSION_DAY  ADMISSION_YEAR ;
```

- (b) Write an ASSIGNMENT statement to create a new variable **ADMISSION_DATE**

2. Use PROC PRINT to display the observations and variables in the data set.

You may also include this **Format** statement to display a "real" date

```
Format  admission_date  MMDDYY8. ;
```

3. Write another DATA step and read the variables STUDENT, START_DATE, and GRAD_DATE.

START_DATE represents the date a student started degree

GRAD_DATE represents the date a student completed degree

```
DATA CLASS;
```

```
INPUT  STUDENT $
```

```
      @6 START_DATE MMDDYY8.
```

```
      @14 GRAD_DATE MMDDYY8. ;
```

```
/*count!...in this example the starting positions for the
   date(s) are in col. 6 and 14 */
```

```
DATALINES;
```

```
1180 9/14/87 6/5/91
```

```
1069 9/8/87 5/20/91
```

```
1200 8/24/82 5/29/86
```

```
;
```

- (a) Create a new variable Number__Years which represents the number of years the student attended school.

4. Use PROC PRINT to display the observations and variables in this data set.

```
PROC PRINT;
```

```
FORMAT START_DATE GRAD_DATE MMDDYY8. ;
```

```
/* Format statement associates the MMDDYY8. date format with the
   variables START_DATE and GRAD_DATE. */
```

```
RUN;
```

EXERCISE 5 (computer-assisted)

(1) Write a DATA step which creates a SAS data set GRANTS

```
DATA GRANTS ;  
    INPUT  INSTITUT $   @7 AMOUNT  COMMA7. ;  
DATALINES;  
NICHD 120,000  
NIAID 220,000  
NCI   180,000  
;
```

1a) Write a RETAIN statement which initializes a new variable **TOTAL1**.

1b) Write an ASSIGNMENT statement.

Accumulate totals(that is, 'a running' sum) for the variable AMOUNT, and assign the totals to the new variable **TOTAL1**.

1c) Write an SUM statement.

Accumulate totals(that is, 'a running' sum) for the variable AMOUNT, and allow a new variable, **TOTAL2**, to retain the 'running' sum or totals for the variable AMOUNT.

(2) Use PROC PRINT to list the observations and variables in this data set.

```
PROC PRINT ;  
    FORMAT AMOUNT TOTAL1 TOTAL2 COMMA7. ;  
    /* Format statement associates the COMMA7. format with the variables  
    AMOUNT, TOTAL1 and TOTAL2 */  
RUN;
```


MODULE 5: CONDITIONAL STATEMENTS

- IF-THEN Statement
- Comparison, Logical and IN Operators
- IF-THEN/ELSE statements
- Subsetting IF Statement
- IF-THEN/DELETE Statement
- OUTPUT Statement

6. IF-THEN STATEMENT

Purpose: Used in the DATA step to conditionally perform statements.

General Form: IF condition THEN statement;

|
 (A)

|
 (B)

(A) condition	an expression that SAS evaluates as true or false
---------------	---

(B) statement an executable statement

If the condition is true then SAS executes the statement.
If the condition is false SAS ignores the statement.

Samples:

- (1) IF SEX = 'F' THEN COUNT + 1;
- (2) IF AGE = 10 THEN GROUP='A';
- (3) IF SEX = ' ' THEN MISSING + 1;
- (4) IF AGE = . THEN AGE=0;

7. COMPARISON, LOGICAL AND IN OPERATORS

7.1 Comparison Operators

KEYWORD	SYMBOL	MEANING
EQ	=	equal to
LT	<	less than
GT	>	greater than
NE	^=	not equal to
LE	<=	less than or equal to
GE	>=	greater than or equal to

Samples:

- (1) IF AGE >= 50 THEN AGE = 39;
- (2) IF SEX EQ 'M' THEN MALE = 1;
- (3) IF TEMP LT 32 THEN WEATHER = 'COLD';

7.2 Logical Operators

KEYWORD	SYMBOL	MEANING															
AND	&	<p>If both expressions linked by AND are true, then the result is true; otherwise, the result is false.</p> <table><tr><th>expression1</th><th>expression2</th><th>result</th></tr><tr><td>T</td><td>T</td><td>T</td></tr><tr><td>T</td><td>F</td><td>F</td></tr><tr><td>F</td><td>T</td><td>F</td></tr><tr><td>F</td><td>F</td><td>F</td></tr></table>	expression1	expression2	result	T	T	T	T	F	F	F	T	F	F	F	F
expression1	expression2	result															
T	T	T															
T	F	F															
F	T	F															
F	F	F															
OR		<p>If either of the expressions linked by OR is true then the result is true; otherwise, the result is false.</p> <table><tr><th>expression1</th><th>expression2</th><th>result</th></tr><tr><td>T</td><td>T</td><td>T</td></tr><tr><td>T</td><td>F</td><td>T</td></tr><tr><td>F</td><td>T</td><td>T</td></tr><tr><td>F</td><td>F</td><td>F</td></tr></table>	expression1	expression2	result	T	T	T	T	F	T	F	T	T	F	F	F
expression1	expression2	result															
T	T	T															
T	F	T															
F	T	T															
F	F	F															
NOT ^		<p>The result of putting NOT in front of an expression whose value is false is true. The result is true if the value of the expression is false.</p> <table><tr><th>expression</th><th>result</th></tr><tr><td>T</td><td>F</td></tr><tr><td>F</td><td>T</td></tr></table>	expression	result	T	F	F	T									
expression	result																
T	F																
F	T																

Samples: Using logical operators in conditional statements

(1) IF STATE = 'NC' OR STATE = 'SC' OR STATE = 'VA' THEN REGION = 'E';

(2) IF STATE = 'NC' AND CITY = 'RALEIGH' THEN STCODE = 3;

(3) IF SCORE >= 80 AND SCORE < 90 THEN GRADE = 'B';

is equivalent to

IF 80 <= SCORE < 90 THEN GRADE = 'B';

(4) IF AGE > 30 AND MARITAL = 'M' THEN CODE = '5';

(5) IF SEX EQ 'F' & AGE LT 14 THEN GROUP = 2;

7.3 IN Operator

Purpose: The IN operator is used to determine whether a variable's value is among a list of values.

General Form: IN (value1, value2, ... , valuen)

Samples:

(1) IF FIPSCODE IN (6,16,32,41,53) THEN REGION = 'SE';

(2) IF STATE IN ('NC','SC','VA') THEN REGION = 'E';

8. IF-THEN/ELSE STATEMENT

Purpose: Used in the DATA step to conditionally perform statements.

General Form: **IF** condition **THEN** statement1;
 ELSE IF condition **THEN** statement2;
 ELSE IF condition **THEN** statement3;
 ELSE statement4;

If a condition is true, SAS executes **statement1**. If a condition is false, SAS ignores **statement1**. SAS continues this same logical process for the other IF-THEN/ELSE series.

Notice the final ELSE statement(above).

Sometimes the last ELSE statement in a series is a little different, containing just a statement, with no IF or THEN. An ELSE of this kind becomes a default which is automatically executed for all observations failing to satisfy any of the previous IF-THEN/ELSE statements. You can only have one of these statements, and it must be the last in the IF-THEN/ELSE series.

Samples:

(1) **IF** 70 <= TEST <= 100 **THEN** SCORE = 'PASS';
 ELSE SCORE = 'FAIL';

One of the most common uses of IF-THEN or IF-THEN/ELSE statements is for grouping observations. Perhaps a variable has too many different values and you want to run an analysis based on specific groups of interest.

(2) **IF** 0 <= AGE <= 20 **THEN** AGEGRP = 1;
 ELSE IF 20 < AGE <= 40 **THEN** AGEGRP = 2;
 ELSE IF 40 < AGE <= 60 **THEN** AGEGRP = 3;
 ELSE IF AGE > 60 **THEN** AGEGRP = 4;
 ELSE AGEGRP = 0;

Advantages of the IF-THEN/ELSE over a simple series of IF-THEN:

- 1) It is more efficient, using less computer time; once an observation satisfies a condition, SAS skips the rest of the series.
- 2) **ELSE** logic ensures that your groups are mutually exclusive so you don't accidentally have an observation fitting into more than one group.

EXERCISE 6 (a written exercise)

(1) Suppose we have the data lines below:

FNAME	SEX	AGE	HT	WT
PAUL	M	27	72	140
JENNIFER	F	28	64	135
RENEE	F	35	54	128
MANUEL	M	35	60	125
TONI	M	32	68	130

Write IF statements:

(a) assign a value of 'NO' to a new variable named ALLOWED if the age is at least 31.

(b) change the first name TONI to TONY.

(c) assign a value of 'A' to a new variable GROUP if the first name is PAUL, RENEE or MANUEL.

(d) create a new variable W according to the value of WT. If WT is between 10 and 130, set W to '130 OR LESS'; otherwise, set it to 'OVER 130'.

(2) Rewrite any of the following IF statements if they are not valid:

(a) IF SEX EQ 'M' THEN CODE EQ 1;

(b) IF VEHID IN (1,3,4) THEN TRUCK + 1 ;

(c) IF (STATE = 'ME' AND CITY = 'PORTLAND') AND
(STATE = 'NJ' AND CITY = 'TRENTON') THEN COUNT + 1;

(d) IF MEAN (OF DRUG1-DRUG5) > 45 THEN LEVEL=HIGH;

9. SUBSETTING IF STATEMENT

Purpose: To select a subset of observations.

General Form: IF condition;

If the **condition is true**, SAS **continues processing** the following statements in the DATA step for the current observation.

If the **condition is false**, then no further statements are processed for that observation; that observation is not added to the SAS data set being created; and SAS moves on to the next observation. Therefore, **control is returned to the beginning of the DATA step**.

You can think of the subsetting IF as a kind of on-off switch. If the condition is true, then the switch is on and the observation is processed. If the condition is false, then that observation is turned off

Sample:

```
DATA MALE;
  INPUT FNAME $ SEX $ AGE HT WT;
  IF SEX = 'M';
DATALINES;
PAUL    M 27 72 140
JENNIFER F 28 64 135
RENEE   F 35 54 128
MANUEL  M 35 60 125
TONY    M 32 68 130
;
```

```
PROC PRINT;
RUN;
```

Sample Output:

OBS	FNAME	SEX	AGE	HT	WT
1	PAUL	M	27	72	140
2	MANUEL	M	35	60	125
3	TONY	M	32	68	130

10. IF-THEN/DELETE STATEMENT

Purpose: To select a subset of observations.

General Form: IF *condition* THEN DELETE;

$$(A)$$

(A) **DELETE** stops processing the current observation when the condition is true. The observation is not written to the SAS data set. Control is returned to the beginning of the DATA step.

While the subsetting IF statement tells SAS which observations to include, the **DELETE** statement tells SAS which observation to exclude.

Generally, you use the **DELETE** statement when it is easier to specify a *condition* for excluding observations. (Easier...meaning you would have to do a lot less typing to specify the *condition*)

Sample 1: Using the IF-THEN/DELETE Statement

```
DATA MALES;
  INPUT FNAME$ SEX$ AGE HT WT;
  IF SEX = 'F' THEN DELETE;
DATALINES;
PAUL    M 27 72 140
JENNIFER F 28 64 135
RENEE   F 35 54 128
MANUEL  M 35 60 125
TONY    M 32 68 130
;
```

```
PROC PRINT;
RUN;
```

Sample Output 1:

OBS	FNAME	SEX	AGE	HT	WT
1	PAUL	M	27	72	140
2	MANUEL	M	35	60	125
3	TONY	M	32	68	130

Sample 2: Using the IF-THEN/DELETE Statement

```
DATA PRODUCT;  
  INPUT DEPT $ UNITS COST ;  
  IF UNITS <= 0 OR COST <= 0 THEN DELETE;  
  UNITCOST = COST/UNITS;  
DATALINES;  
CCB 10 525.00  
LSM 50 -6.00  
LAS 5 100.00  
PCB 0 3.00  
OD 1 15.00  
;  
  
PROC PRINT;  
RUN;
```

Sample Output 2:

OBS	DEPT	UNITS	COST	UNITCOST
1	CCB	10	525	52.5
2	LAS	5	100	20.0
3	OD	1	15	15.0

11. OUTPUT STATEMENT

Purpose: When an OUTPUT statement is used in a DATA step, there is no automatic output at the end of the DATA step. Rather, the OUTPUT statement controls when the observations are written to the output SAS data set. After an OUTPUT statement, control does not return to the start of the DATA step; SAS continues processing any other statements in the step.

General Form: OUTPUT sasdsname;

|
(A)

(A) sasdsname SAS data set(s) must be named in the DATA statement.

Sample 1: Using the OUTPUT statement to write to multiple SAS data sets

```
DATA FEMALES MALES ;
  INPUT FNAME $ SEX $ AGE HT WT;
  IF SEX = 'M' THEN OUTPUT MALES;
  ELSE IF SEX = 'F' THEN OUTPUT FEMALES;
DATALINES;
PAUL M 27 72 140
JENNIFER F 28 64 135
RENEE F 35 54 128
MANUEL M 35 60 125
TONY M 32 68 130
;
PROC PRINT DATA=FEMALES;
  TITLE 'FEMALES';
PROC PRINT DATA=MALES;
  TITLE 'MALES'; RUN;
```

Sample Output 1:

FEMALES					
OBS	FNAME	SEX	AGE	HT	WT
1	JENNIFER	F	28	64	135
2	RENEE	F	35	54	128
MALES					
OBS	FNAME	SEX	AGE	HT	WT
1	PAUL	M	27	72	140
2	MANUEL	M	35	60	125
3	TONY	M	32	68	130

Sample 2: OUTPUT statement to generate several observations from one

Given the following variables and datalines

IDNUM	SCORE1	SCORE2	SCORE3
21573148	82	91	78
13429576	91	94	88

We will use the OUTPUT statement to generate several observations from one

In this case no SAS data set is specified in the OUTPUT statement, therefore the observation is written to the SAS data set named in the DATA statement.

```
DATA OUTPUT2 ;  
  INPUT IDNUM $ SCORE1 - SCORE3 ;  
  TEST = 1 ;  
  SCORE = SCORE1 ;  
  OUTPUT ;  
  TEST = 2 ;  
  SCORE = SCORE2 ;  
  OUTPUT ;  
  TEST = 3 ;  
  SCORE = SCORE3 ;  
  OUTPUT ;  
  DROP SCORE1 - SCORE3 ;  
DATALINES ;  
21573148 82 91 78  
13429576 91 94 88  
;  
  
PROC PRINT ;  
RUN ;
```

Sample Output 2:

OBS	IDNUM	TEST	SCORE
1	21573148	1	82
2	21573148	2	91
3	21573148	3	78
4	13429576	1	91
5	13429576	2	94
6	13429576	3	88

12. LENGTH STATEMENT

Purpose: To define the number of bytes used to store values of variables in the SAS data set.

General Form:

LENGTH	variable(s)	[\$]	length...	[DEFAULT=n] ;
	(A)		(B)	(C)

- | | |
|------------|--|
| (A) \$ | indicates the preceding variable or variables are character variables |
| (B) Length | <p>In the Windows/Mac/Unix environments this constant represents a range from 3 to 8 for numeric variables and 1 to 32,767 bytes for character variables.</p> <p>In the OS/390 environment this constant represents a range from 2 to 8 for numeric variables and 1 to 32,767 bytes for character variables</p> |
| (C) n | changes the default number of bytes used for storing the values of newly created numeric variables from 8 (the default length) to the value of n(n can range from 2 to 8 or 3 to 8, depending on the host system) |

The following table shows the largest integer represented for the corresponding **Length in Bytes**. Limits shown in this table are for the **Windows/Mac/Unix** environments. Limits are different for the OS/390 environment.

Length in Bytes	Largest Integer Represented	Exponential notation
3	8,192	2 to the 13th power
4	2,097,152	2 to the 21st power
5	536,870,912	2 to the 29th power
6	137,438,953,472	2 to the 37th power
7	35,184,372,088,832	2 to the 45th power
8	9,007,199,254,740,992	2 to the 53rd power

Example:

If you know that a numeric variable always has values between 0 and 100, you can use a length of 3 to store the number and save space on storage.

Note: You can safely use the LENGTH statement when the value for the variable are integers. For non-integers (fractional values) it is highly recommended that you **do not** use the LENGTH statement.

Advantages:

- (1) Save space on storage
- (2) Saves time in reading and writing the data.
- (3) Changes SAS defaults for storing the values

Sample 1:

The length of a character variable is determined by the value it takes the first time it appears in the DATA step.

```
DATA BYTES;
  INPUT X ;
  IF X = 1 THEN A = 'NO' ; /* the first time the variable A appears in the DATA step*/
  ELSE A = 'YES' ;
DATALINES;
2
4
1
;

PROC PRINT ;
RUN;
```

Sample output 1:

OBS	X	A
1	2	YE
2	4	YE
3	1	NO

Sample 2:

To avoid the problem shown in **Sample 1**, use the LENGTH statement to give A a length of 3.

```
DATA BYTES;
  LENGTH A $ 3 ;
  INPUT  X ;
  IF X = 1 THEN A = 'NO' ;
  ELSE A = 'YES' ;
DATALINES;
2
4
1
;

PROC PRINT ;

PROC CONTENTS;
RUN;
```

Sample Output 2:

OBS	A	X
1	YES	2
2	YES	4
3	NO	1

Proc content output:

CONTENTS PROCEDURE

Data Set Name:	WORK.BYTES	Observations:	3
Member Type:	DATA	Variables:	2
Engine:	V8	Indexes:	0
Created:	10:10 Monday, September 18, 2000	Observation Length:	11
Last Modified:	10:10 Monday, September 18, 2000	Deleted Observations:	0
Protection:		Compressed:	NO
Data Set Type:		Sorted:	NO
Label:			

-----Alphabetic List of Variables and Attributes-----

#	Variable	Type	Len	Position
1	A	Char	3	0
2	X	Num	8	3

Sample 3:

```
DATA DEFAULT;  
  LENGTH NAME $ 10 DEFAULT = 3 ; /* 3 bytes to store integer <=8,192 */  
  INPUT NAME $ SCORE ;  
DATALINES;  
JASON      174  
ROSA       195  
FRITZ      188  
KATHERINE  199  
;  
  
PROC PRINT ;  
  
PROC CONTENTS;  
RUN;
```

The LENGTH statement sets the length of the character variable NAME to 10 and changes the default number of bytes used for storing numeric variables from 3 to 8 bytes.

Sample Output 3:

OBS	NAME	SCORE
1	JASON	174
2	ROSA	195
3	FRITZ	188
4	KATHERINE	199

Proc Contents output:

CONTENTS PROCEDURE			
Data Set Name:	WORK.DEFAULT	Observations:	4
Member Type:	DATA	Variables:	2
Engine:	V8	Indexes:	0
Created:	10:10 Monday, September 18, 2000	Observation Length:	13
Last Modified:	10:10 Monday, September 18, 2000	Deleted Observations:	0
Protection:		Compressed:	NO
Data Set Type:		Sorted:	NO
Label:			

-----Alphabetic List of Variables and Attributes-----

#	Variable	Type	Len	Pos
1	NAME	Char	10	0
2	SCORE	Num	3	10

EXERCISE 7 (computer-assisted exercise)

It is a good practice to write-in your SAS statements in the blank spaces provided before you enter the SAS program in the computer.

Suppose we have the following data lines which correspond to the variables FNAME, SEX, AGE, HT, and WT

PAUL	M	27	72	140
JENNIFER	F	28	64	135
RENEE	F	35	54	128
MANUEL	M	35	60	125
TONY	M	32	68	130

- (1) Write a DATA step and create a SAS data set **AGE** that includes only those individuals 30 years old or older.

- (2) Write a DATA step and create a SAS data set **MALES** that includes only males over 30.

- (3) Write a third DATA step and create two SAS data sets called **GROUP1** and **GROUP2**.
GROUP1 includes only females over 30
GROUP2 includes only males over 30.

MODULE 6: MANAGING SAS DATA SETS

- PROC SORT
- SET Statement
- MERGE Statement
- DROP = and KEEP = (SAS data set options)
- WHERE Statement

13. PROC SORT

Purpose: PROC SORT rearranges the observations in a SAS data set. PROC SORT does not generate a printed output.

General Form:

```
PROC SORT DATA = sasdsname1 OUT = sasdsname2 ;
      BY variables ;
```

(C) (A) (B)

(A) `sasdsname1` names the SAS data set to be sorted...if you do not specify the `DATA=` option, then SAS will use the most recently created SAS data set

(B) `sasdsname2` names the output SAS data set...if you do not specify the `OUT=` option, then SAS will replace the original SAS data set with the newly sorted version

(C) variables list any number of variables in the BY statement. By default, the observations are sorted in ascending order of the BY variables. You can sort the observations in descending order using the keyword **DESCENDING** before each variable you want sorted in descending order.

There are several terms used in BY processing:

by-variable	variable in a BY statement
-------------	----------------------------

by-value	value of a by-variable
----------	------------------------

by-group	observations with the same by-values
----------	--------------------------------------

Sample 1: PROC SORT with the OUT = option

```
PROC SORT DATA = ORIG OUT = SORTORIG ;  
  BY GRADE ;
```

SAS data set ORIG

OBS	GRADE	NAME
1	A	SUE
2	C	JANE
3	B	BILL
4	A	MATT
5	D	DAVE

SAS data set SORTORIG

OBS	GRADE	NAME
1	A	SUE
2	A	MATT
3	B	BILL
4	C	JANE
5	D	DAVE

Sample 2: PROC SORT without the OUT = option

```
PROC SORT DATA = ORIG ;  
  BY GRADE ;
```

SAS data set ORIG

BEFORE THE SORT

OBS	GRADE	NAME
1	A	SUE
2	C	JANE
3	B	BILL
4	A	MATT
5	D	DAVE

SAS data set ORIG

AFTER THE SORT

OBS	GRADE	NAME
1	A	SUE
2	A	MATT
3	B	BILL
4	C	JANE
5	D	DAVE

Sample 3: PROC SORT with 2 by-variables

```
PROC SORT DATA = ORIG ;  
  BY GRADE NAME ;
```

SAS data set ORIG

BEFORE THE SORT

OBS	GRADE	NAME
1	A	SUE
2	C	JANE
3	B	BILL
4	A	MATT
5	D	DAVE

SAS data set ORIG

AFTER THE SORT

OBS	GRADE	NAME
1	A	MATT
2	A	SUE
3	B	BILL
4	C	JANE
5	D	DAVE

14. SET STATEMENT

Purpose: The SET statement instructs SAS to read observations from one or more existing SAS data sets rather than from data lines. This allows you to read a SAS data set so you can add new variables, create a subset, or otherwise modify the SAS data set.

General Form: DATA new-sasdataset;
 SET sasdsname1 sasdsname2 . . . sasdsnamen ;
 |
 (A)

(A) sasdsnamen A SET statement can read any number of SAS data sets.

A new program data vector is defined that contains all of the variables found in the existing SAS data sets plus any new variables created with other SAS statements. By default, observations are read from the first data set, then the second, and so on through all the data sets. The SET statement is executed once for each observation in the SAS data sets.

Sample:

```
DATA NEW ;  
  SET OLD ;  
  RATIO = HT/WT ;
```

SAS data set OLD

FNAME	SEX	AGE	HT	WT
PAUL	M	27	72	140
JENNIFER	F	28	64	135
RENEE	F	35	54	128
MANUEL	M	35	60	125
TONY	M	32	68	130

SAS data set NEW

FNAME	SEX	AGE	HT	WT	RATIO
PAUL	M	27	72	140	0.514
JENNIFER	F	28	64	135	0.474
RENEE	F	35	54	128	0.422
MANUEL	M	35	60	125	0.480
TONY	M	32	68	130	0.523

14.1 Concatenating SAS Data Sets

Purpose: Observations from one data set are stacked with observations from the other data set. This is useful when you want to combine SAS data sets with all or most of the same variables but different observations.

General Form: DATA new-sasdataset;
 SET sasdsname1 sasdsname2 . . . sasdsnamen ;

Sample 1:

```
DATA COMMON;  
  SET NWEST SWEST ;
```

SAS data set NWEST

OBS	STATE	POP
1	WA	3.4
2	OR	2.1

SAS data set SWEST

OBS	STATE	POP
1	NM	1.1
2	AZ	1.8

SAS data set COMMON

OBS	STATE	POP
1	WA	3.4
2	OR	2.1
3	NM	1.1
4	AZ	1.8

Sample 2: Concatenating

```
DATA UNCOMMON;  
  SET OLD1 OLD2;
```

SAS data set OLD1

OBS	ID	X	Y	Z
1	1	1	9	2
2	2	1	3	6
3	3	1	4	8

SAS data set OLD2

OBS	A	B	C	ID
1	2	2	2	4
2	3	5	5	5
3	2	9	2	6
4	2	3	4	7

SAS data set UNCOMMON

OBS	ID	X	Y	Z	A	B	C
1	1	1	9	2	.	.	.
2	2	1	3	6	.	.	.
3	3	1	4	8	.	.	.
4	4	.	.	.	2	2	2
5	5	.	.	.	3	5	5
6	6	.	.	.	2	9	2
7	7	.	.	.	2	3	4

14.2 Interleaving SAS Data Sets

Purpose: Interleaving combines individual sorted SAS data sets into one sorted data set.
If you have data sets that are already sorted by some common variable, then simply stacking the SAS data sets may unsort the data sets. In such a case all you need to do is use a BY statement with your SET statement.

General Form: DATA new-sasdataset;
 SET sasdsname1 sasdsname2 . . . sasdsnamen;
 BY variables ;

As the SET statement is executed, SAS checks the current observation in each data set and determines which observation to process by examining the values of the BY variable(s).

Sample 1: Interleaving

```
DATA INTERLV1;  
  SET NWEST SWEST;  
  BY STATE;
```

SAS data set NWEST

OBS	STATE	POP
1	NM	1.1
2	WA	3.4

SAS data set SWEST

OBS	STATE	POP
1	AZ	1.8
2	OR	2.1

SAS data set INTERLV1

OBS	STATE	POP
1	AZ	1.8
2	NM	1.1
3	OR	2.1
4	WA	3.4

Sample 2: Interleaving - without multiple By- values

```
DATA INVERLV2 ;  
  SET DATA1 DATA2 ;  
  BY ID ;
```

SAS data set DATA1

OBS	ID	TREAT1
1	1	A1
2	2	A2
3	5	A3

SAS data set DATA2

OBS	ID	TREAT2
1	3	B1
2	4	B2
3	6	B3

SAS data set INTERLV2

OBS	ID	TREAT1	TREAT2
1	1	A1	.
2	2	A2	.
3	3	.	B1
4	4	.	B2
5	5	A3	.
6	6	.	B3

Sample 3: Interleaving with multiple 'By-values'

```
DATA INVERLV3;  
  SET DATA1 DATA2;  
  BY NUM;
```

SAS data set DATA1

OBS	NUM	TREAT1
1	1	A1
2	2	A2
3	2	A3
4	3	A4

SAS data set DATA2

OBS	NUM	TREAT2
1	2	B1
2	3	B2
3	3	B3

SAS data set INTERLV3

OBS	NUM	TREAT1	TREAT2
1	1	A1	.
2	2	A2	.
3	2	A3	.
4	2	.	B1
5	3	A4	.
6	3	.	B2
7	3	.	B3

15. MERGE STATEMENT

Purpose: The MERGE statement is used to join corresponding observations from two or more SAS data sets.

General Form: DATA new-sasdataset;
MERGE sasdsname1 sasdsname2 . . . sasdsnamen ;
BY variables ;

15.1 One-To-One Merging is done without a BY statement...joins observations by position

Purpose: A one-to-one merge combines the first observation from all the data sets in the MERGE statement into the first observation in the new data set, the second observation from all the data sets into the second observation in the new data set, and so on. The new data set has the same number of observations as the largest data set in the MERGE statement.

General Form: DATA new-sasdataset;
MERGE sasdsname1 sasdsname2 . . . sasdsnamen ;

Sample 1: One-to-One Merge...joins observations by position(e.g. by observation number)

```
DATA NEW;  
  MERGE NAMES SURVY;
```

SAS data set NAMES

OBS	NAME	SEX
1	PAUL	M
2	JENNIFER	F
3	RENEE	F
4	MANUEL	M
5	TONY	M

SAS data set SURVY

OBS	AGE	HEIGHT	WEIGHT
1	27	72	140
2	28	64	135
3	35	54	128
4	35	60	125
5	32	68	130

SAS data set NEW

OBS	NAME	SEX	AGE	HEIGHT	WEIGHT
1	PAUL	M	27	72	140
2	JENNIFER	F	28	64	135
3	RENEE	F	35	54	128
4	MANUEL	M	35	60	125
5	TONY	M	32	68	130

Sample 2: One-to-One Merge...joins observations by position(e.g. by observation number)

```
DATA NEW;  
  MERGE NAMES SURVY;
```

SAS data set NAMES

OBS	NAME	SEX
1	PAUL	M
2	JENNIFER	F
3	RENEE	F
4	MANUEL	M
5	TONY	M

SAS data set SURVY

OBS	AGE	HEIGHT	WEIGHT
1	27	72	140
2	28	64	135
3	35	54	128
4	35	60	125

SAS data set NEW

OBS	NAME	SEX	AGE	HEIGHT	WEIGHT
1	PAUL	M	27	72	140
2	JENNIFER	F	28	64	135
3	RENEE	F	35	54	128
4	MANUEL	M	35	60	125
5	TONY	M	.	.	.

Sample 3: One-to-One Merge...joins observations by position(e.g. by observation number)

```
DATA NEW;  
  MERGE NAMES SURVY;
```

SAS data set NAMES

OBS	NAME
1	PAUL
2	JENNIFER
3	RENEE
4	MANUEL
5	TONY

SAS data set SURVY

OBS	SEX	AGE	HEIGHT	WEIGHT
1	F	28	64	135
2	F	35	54	128
3	M	35	60	125
4	M	32	68	130

SAS data set NEW

OBS	NAME	SEX	AGE	HEIGHT	WEIGHT
1	PAUL	F	28	64	135
2	JENNIFER	F	35	54	128
3	RENEE	M	35	60	125
4	MANUEL	M	32	68	130
5	TONY

Sample 4: One-to-One Merge...joins observations by position(e.g. by observation number)

```
DATA NEW;  
  MERGE NAMES SURVY;
```

SAS data set NAMES

OBS	NAME	SEX
1	PAUL	M
2	JENNIFER	F
3	RENEE	F
4	MANUEL	M
5	TONY	M

SAS data set SURVY

OBS	AGE	HEIGHT	NAME
1	27	72	SMITH
2	28	64	JONES
3	35	54	PETERS
4	35	60	RUIZ
5	32	68	ANGELO

SAS data set NEW

OBS	NAME	SEX	AGE	HEIGHT
1	SMITH	M	27	72
2	JONES	F	28	64
3	PETERS	F	35	54
4	RUIZ	M	35	60
5	ANGELO	M	32	68

15.2 Match Merging

Purpose: Match merging combines observations from two or more SAS data sets based on the values of one or more common variables specified in the BY statement.

The SAS data sets must be sorted by the BY-variable(s).

General Form: DATA new-sasdataset;
MERGE sasdsname1 sasdsname2 . . . sasdsnamen ;
BY variable(s);

Sample 1:

```
DATA NEW ;  
MERGE NAMES SURVY ;  
BY NAME ;
```

SAS data set NAMES

OBS	NAME	SEX
1	JENNIFER	F
2	MANUEL	M
3	PAUL	M
4	RENEE	F
5	TONY	M

SAS data set SURVY

OBS	AGE	HEIGHT	NAME
1	28	64	JENNIFER
2	35	60	MANUEL
3	27	72	PAUL
4	35	54	RENEE
5	32	68	TONY

SAS data set NEW

OBS	NAME	SEX	AGE	HEIGHT
1	JENNIFER	F	28	64
2	MANUEL	M	35	60
3	PAUL	M	27	72
4	RENEE	F	35	54
5	TONY	M	32	68

Sample 2: Match Merge

```
DATA NEW;  
  MERGE NAMES SURVY;  
  BY NAME;
```

SAS data set NAMES

OBS	NAME	SEX
1	JENNIFER	F
2	MANUEL	M
3	PAUL	M
4	RENEE	F
5	TONY	M

SAS data set SURVY

OBS	AGE	HEIGHT	NAME
1	28	64	JENNIFER
2	35	60	MANUEL
3	35	54	RENEE
4	32	68	TONY

SAS data set NEW

OBS	NAME	SEX	AGE	HEIGHT
1	JENNIFER	F	28	64
2	MANUEL	M	35	60
3	PAUL	M	.	.
4	RENEE	F	35	54
5	TONY	M	32	68

Sample 3: Match Merge - multiple BY-value(s) or “repeat” of BY-value(s)

```
DATA NEW;  
  MERGE NAMES SURVY;  
  BY NAME;
```

SAS data set NAMES

OBS	NAME	SEX
1	JENNIFER	F
2	MANUEL	M
3	PAUL	M
4	RENEE	F
5	TONY	M

SAS data set SURVY

OBS	AGE	HEIGHT	NAME
1	28	64	JENNIFER
2	35	60	MANUEL
3	27	72	PAUL
4	35	54	RENEE
5	37	56	RENEE
6	32	68	TONY

SAS data set NEW

OBS	NAME	SEX	AGE	HEIGHT
1	JENNIFER	F	28	64
2	MANUEL	M	35	60
3	PAUL	M	27	72
4	RENEE	F	35	54
5	RENEE	F	37	56
6	TONY	M	32	68

Sample 4: Match Merge - multiple BY-value(s) or “repeat” of BY-value(s)

```
DATA NEW;  
  MERGE NAMES SURVY;  
  BY NAME;
```

SAS data set NAMES

OBS	NAME	SEX
1	JENNIFER	F
2	MANUEL	M
3	PAUL	M
4	RENEE	F
5	RENEE	M
6	RENEE	
7	TONY	M

SAS data set SURVY

OBS	AGE	HEIGHT	NAME
1	28	64	JENNIFER
2	35	60	MANUEL
3	27	72	PAUL
4	35	54	RENEE
5	37	56	RENEE
6	32	68	TONY

SAS data set NEW

OBS	NAME	SEX	AGE	HEIGHT
1	JENNIFER	F	28	64
2	MANUEL	M	35	60
3	PAUL	M	27	72
4	RENEE	F	35	54
5	RENEE	M	37	56
6	RENEE		37	56
7	TONY	M	32	68

NOTE: MERGE statement has more than one data set with repeat of BY values.

16. Using the **DROP =** or the **KEEP =**

Purpose: The **DROP=** specifies variables that SAS should not write to the output data set. If the option is associated with an input data set, the variables are not available for processing. The **DROP=** option can be used in either a DATA or PROC step. The **KEEP=** SAS data set option controls which variables are processed or written to output SAS data sets during a DATA or PROC step. If the option is associated with an input data set, only those variables listed after the **KEEP=** option are available for processing.

General Form: DATA new-sasdataset;
sasdsname(**DROP =** variable1 variable2 ... variablen) ;

or

sasdsname(**KEEP =** variable1 variable2 ... variablen) ;

Sample 1:

```
DATA NEW ;  
  SET MALE(KEEP = NAME AGE) FEMALE(KEEP = NAME HEIGHT) ;
```

SAS data set MALE

OBS	NAME	AGE	HEIGHT
1	PAUL	27	72
2	MANUEL	35	60
3	TONY	32	68

SAS data set FEMALE

OBS	NAME	AGE	HEIGHT
1	JENNIFER	28	64
2	RENEE	35	54

SAS data set NEW

OBS	NAME	AGE	HEIGHT
1	PAUL	27	.
2	MANUEL	35	.
3	TONY	32	.
4	JENNIFER	.	64
5	RENEE	.	54

Sample 2:

```
DATA NEW;  
  MERGE NAMES SURVY(DROP = NAME);
```

SAS data set NAMES

OBS	NAME	SEX
1	PAUL	M
2	JENNIFER	F
3	RENEE	F
4	MANUEL	M
5	TONY	M

SAS data set SURVY

OBS	AGE	HEIGHT	NAME
1	27	72	SMITH
2	28	64	JONES
3	35	54	PETERS
4	35	60	RUIZ
5	32	68	ANGELO

SAS data set NEW

OBS	NAME	SEX	AGE	HEIGHT
1	PAUL	M	27	72
2	JENNIFER	F	28	64
3	RENEE	F	35	54
4	MANUEL	M	35	60
5	TONY	M	32	68

17. WHERE STATEMENT

Purpose: The WHERE statement allows the user to select a subset of observations satisfying one or more conditions from an existing SAS data set. This statement can be used in either a PROC or a DATA step. The WHERE statement may only be used in a DATA step with a SET, MERGE, or UPDATE statement.

General Form: WHERE *where-expression* ;

|
(A)

(A) *where-expression* a valid arithmetic or logical expression

Samples:

(1) WHERE SCORE > 50 ;

(2) WHERE DATE >= '01JAN89'D ;

(3) WHERE STATE = 'MISSISSIPPI' ;

(4) WHERE RACE = • ;

/*combines two conditions by finding observations that satisfy either condition.*/
(5) WHERE CITY = 'NEW YORK' OR CITY = 'NEWARK' ;

/*combines two conditions by finding observations that satisfy both conditions.*/
(6) WHERE SKILL EQ 'SAS' and YEARS EQ 4 ;

(7) WHERE (NUM <2 OR NUM >4) ;

17.1 WHERE Operators

The following operators can be used only with **WHERE** processing.

17.1.1 BETWEEN—AND Operator

Purpose: Selects observations based on a range of values.

General Form: **WHERE** variable **BETWEEN** value1 **AND** value2 ;

value1 and **value2** are constants or expressions which specify the limits of the range of values...**value1** and **value2** are included in the range.

Samples:

(1) **WHERE SALES BETWEEN 90 AND 100 ;**

You can also combine the **NOT** operator with the **BETWEEN—AND** operator to select values that fall outside the range.

(2) **WHERE SALES NOT BETWEEN 90 AND 100 ;**

17.1.2 CONTAINS Operator

Purpose: Select observations that contain the character string specified in the **WHERE** expression.

General Form: **WHERE** variable **CONTAINS** 'string' ;

or

WHERE variable ? 'string' ;

The position of the character string in the variable's value does not matter. The **CONTAINS** operator distinguishes between uppercase and lowercase characters when making comparisons.

Sample: **WHERE** LASTNAME ? 'Mc' ;

17.1.3 IS NULL or IS MISSING Operator

Purpose: Selects all observations in which the value of a variable is missing.

General Form: WHERE variable IS NULL ;

or

WHERE variable IS MISSING ;

Sample:

```
DATA NOTNULL ;  
    INPUT NAME $ 1-8 AGE 11-12 ;  
DATALINES;  
    .      27  
JENNIFER  28  
RENEE     35  
MANUEL    .  
TONY      32  
;  
  
PROC PRINT ;  
    WHERE NAME IS NOT NULL and AGE IS NOT MISSING ;  
RUN;
```

Sample Output:

OBS	NAME	AGE
2	JENNIFER	28
3	RENEE	35
4	TONY	32

17.1.4 SOUNDS-LIKE operator in a WHERE clause

Purpose: Selects observations that contain a spelling variation of the word or words specified in the **WHERE** expression.

General Form: WHERE variable =* 'string' ;

Sample:

```
DATA EDITNAME ;  
    INPUT NAME $ ;  
DATALINES ;  
ALAN  
ALLEN  
ALLAN  
DAVE  
JOSEPH  
;  
  
PROC PRINT ;  
    WHERE NAME =* 'ALAN' ;  
RUN;
```

Sample Output:

OBS	NAME
1	ALAN
2	ALLEN
3	ALLAN

17.1.5 LIKE Operator

Purpose: Selects observations with character values matching a specified pattern.

+

General Form: **WHERE** variable **LIKE** 'string1%string2' ;
 WHERE variable **LIKE** 'string1_string2' ;

There are two special character available for specifying patterns:

% The percent sign tells SAS that any number of characters can occupy that position.

_ The underscore sign tells SAS to match one character in the value for each underscore character.

Sample:

```
DATA SELECT ;  
  INPUT NAMES $ ;  
DATALINES;  
DIANA  
DIANE  
DIANNA  
DIANTHUS  
DYAN  
;  
  
PROC PRINT ;  
  WHERE NAME LIKE 'D_AN' or NAME LIKE 'DIA%A' ;  
RUN;
```

Sample Output:

OBS	NAMES
1	DIANA
3	DIANNA
5	DYAN

EXERCISE 8 (computer-assisted)

Write one SAS program and include the following DATA and PROC steps:

- (1) Creates a SAS data set **Maryland**.
Create a second SAS data set **Virginia**.

Each SAS data set represents sample traffic violations in Maryland and Virginia.

SAS data set Maryland

State	Number_Accidents	Violation_Code
MD	118	25
MD	120	10
MD	123	30
MD	124	30

SAS data set Virginia

State	Number_Accidents	Violation_Code
VA	454	25
VA	460	15

- (2) Create the SAS data set **CONCAT** by concatenating the two SAS data set Maryland and Virginia. The SAS data set will include all six observations in the order they appear in the data sets **Maryland** and **Virginia** data sets.
- (3) Sort the SAS data set **CONCAT** by *Violation_Code*.
- (4) Creates a SAS data set **CODE** and select only those observations from the SAS data set **CONCAT** where the **Violation_code is 10**.
- (5) Creates a SAS data set **FEES** which includes the variables Violation_Code and FEE. The resultant SAS data set **FEES** is displayed below.

SAS Data Set FEES

Violation_Code	FEE
10	75
15	45
25	60
30	80

- (6) Create a SAS data set and merge the data sets **CONCAT** and **FEES** so that the variable FEE appears on each of **CONCAT**'s observations matching the Violation_Code.
- (7) Check the SAS log and SAS Output. Is this the desired SAS Output?

EXERCISE 9 (computer-assisted)

Write one SAS program and include the following DATA step and PROC steps:

- (1) A SAS DATA Step creates a SAS data set STUDY.
For instance, the SAS data set STUDY represents the variables ID TREATMENT and RESPONSE. The SAS data set STUDY is shown below.

SAS data set STUDY

OBS	ID	TREATMENT	RESPONSE
1	A	1	35.45
2	A	2	39.80
3	A	3	.
4	B	1	30.53
5	B	2	32.75
6	B	3	37.90
7	C	1	42.25
8	C	2	45.76
9	C	3	.

- (a) Include a WHERE statement in a PROC PRINT step and obtain a listing of the observations where the value of RESPONSE is missing.
- (b) Include a WHERE statement in another PROC PRINT step and obtain a listing of the observations where RESPONSE is between 32 and 38.
- (2) Creates a SAS data set FINAL which includes the observations for TREATMENT 2, where RESPONSE is greater than 35.

**Other Base SAS software
features that may be helpful to
you**

INFILE statement with **FIRSTOBS** = and **OBS** = options

There are a variety of options that can be used with an INFILE statement to control how data are read and to allow the SAS program more control over the input operation.

INFILE statement with **FIRSTOBS** = and **OBS** = options ;

Purpose: To control which records are read from the data file.

General Form: INFILE fileref **FIRSTOBS** = n1 **OBS** = n2 ;
||
(A)(B)

(A)	n1	the record number of the first data line to read
(B)	n2	the record number of the last data line to read

(B) n2 the record number of the last data line to read

Sample external data : filename is DATA1.txt

1	31	62	126
2	20	78	154
3	28	64	128
4	29	96	155
5	21	66	128
6	27	96	265
7	21	68	120
8	42	72	138

Sample Program:

```
FILENAME IN1 'drive:\path for the filename DATA1.txt';
```

```
DATA TEMP1 ;
  INFILE IN1 FIRSTOBS = 3  OBS = 7 ;
  INPUT ID $  AGE  PULSE_1 PULSE_2 ;
```

```
PROC PRINT ;
RUN;
```

Sample Output:

OBS	ID	AGE	PULSE_1	PULSE_2
1	3	28	64	128
2	4	29	96	155
3	5	21	66	128
4	6	27	96	265
5	7	21	68	120

INFILE Statement with MISSOVER option

Purpose: By default, SAS will go to the next data line to read more data if SAS has reached the end of the data line and it still has more variables to read. The MISSOVER option prevents SAS from going to the next data line to read values with LIST input style if it doesn't find values in the current line for all input variables. SAS assigns missing values to any remaining variables.

General Form: **INFILE** fileref **MISSOVER** ;

Sample external data: filename is DATA2

```
87 85
74 80 83
```

Sample Program:

```
FILENAME IN1 'WXYZABC.DATA2' ; /* syntax for SAS program executed on IBM/MVS */
```

```
DATA TEMP2 ;
  INFILE IN1 MISSOVER ;
  INPUT TMP1 - TMP3 ;
```

```
PROC PRINT ;
RUN;
```

Sample Output:

OBS	TMP1	TMP2	TMP3
1	87	85	.
2	74	80	83

INFILE Statement with STOPOVER option

Purpose: To halt program execution in case of missing data so the data can be set right. The STOPOVER option stops processing the DATA step when an INPUT statement using LIST input style reaches the end of the current record without finding values for all variables in the statement. The system variable `_ERROR_` is set to 1, and the offending data line will be printed to the SAS log.

General Form: INFILE fileref **STOPOVER** ;

Sample external data: filename is DATA3

```
87 85
74 80 83
```

Sample Program:

```
FILENAME IN1 'drive:\path for the filename DATA3.txt' ;
DATA TEMP3 ;
  INFILE IN1 STOPOVER ;
  INPUT TMP1 - TMP3 ;
```

Sample SAS Log:

```
1
2      FILENAME IN1 'drive:\path for the filename DATA3.txt' ;
3
4      DATA TEMP3 ;
5          INFILE IN1 STOPOVER ;
6          INPUT TMP1 - TMP3 ;
7
NOTE: The infile IN1 is:
      Dsname=WXYZABC.DATA3,
      Unit=3380,Volume=DSA004,Disp=SHR,Blksize=11475,
      Lrecl=15,Recfm=FB
ERROR: INPUT statement exceeded record length
      INFILE WXYZABC.DATA3 OPTION STOPOVER specified.

RULE:      ----+-----1-----+-----2-----+-----3-----+-----4-----+-----5---
1          87 85
TMP1=87 TMP2=85 TMP3=. _ERROR_=1 _N_=1
NOTE: 1 record was read from the infile IN1.
NOTE: The SAS System stopped processing this step because of
      errors.
NOTE: SAS set option OBS=0 and will continue to check statements.
WARNING:The data set WORK.TEMP3 may be incomplete. When this
step was stopped there were 0 observations and 3 variables.
```

Multiple INFILE and INPUT Statement

Purpose: Read two or more raw data files into a SAS data set with a single DATA step.

Sample external files: Both files contain the same number of records.

External filename is NUM1.txt'

1	2	3
7	8	9
13	14	15

External filename is NUM2.txt'

4	5	6
10	11	12
16	17	18

Sample Program:

```
FILENAME IN1 'drive:\path for filename NUM1.txt' ;  
FILENAME IN2 'drive:\path for filename NUM2.txt' ;
```


```
DATA TEMP5 ;  
  INFILE IN1 ;  
  INPUT A B C ;  
  INFILE IN2 ;  
  INPUT X Y Z ;
```

```
PROC PRINT;  
RUN;
```


Sample Output:

OBS	A	B	C	X	Y	Z
1	1	2	3	4	5	6
2	7	8	9	10	11	12
3	13	14	15	16	17	18


Purpose: Tells SAS that all statements in the **DO** loop are executed repetitively as a unit until a matching **END** statement is encountered.




(A)



(B)



(C)



(D)

END ;

- | | |
|---------------------|--|
| (A) index- variable | names a variable whose value changes in each iteration of the loop. This variable is kept in the SAS data set. |
| (B) start | starting value for the index-variable. |
| (C) stop | ending value for the index-variable. |
| (D) increment | by default the index-variable is incremented by 1 before each iteration of the DO loop. |

```
DATA LOOP1 ;
  DO X=1 TO 10 BY 2 ;
    Y=SQRT(X) ;
    OUTPUT;
  END;
```

Sample Output 1:

81

DO/END Statement

Purpose: Used to execute a group of statements when a condition is met.

General Form 2: IF condition then DO ;

..... other SAS statements

END ;

Sample 2:

```
DATA DOLOOP2 ;  
  INPUT MONTH;  
  IF MONTH LE 0 THEN DO ;  
    BADDATA + 1;  
    MONTH = .;  
  END ;  
DATALINES;  
1  
2  
-4  
3  
0  
;  
PROC PRINT;  
RUN;
```

Sample Output 2:

OBS	MONTH	BADDATA
1	1	0
2	2	0
3	.	1
4	3	1
5	.	2

ARRAY STATEMENT

SAS does have arrays, but they are used in different ways than in traditional programming languages like C, FORTRAN, and BASIC. An array in SAS consists of variables. You use arrays when you want to do the same thing to each variable in the array, and you don't want to write a separate statement for each variable. Arrays are temporary in SAS, existing only for the duration of the DATA step in which they are defined. Arrays provide ways to shorten or simplify your SAS programs.

Purpose: An ARRAY is a convenient way of defining a set of variables to be processed.

General Form:

ARRAY	arrayname	(n)	[\$]	variable1...variablen ;
	(A)	(B)	(C)	(D)

(A) arrayname identifies the group of variables...must not match any of the variable names in your data set or any SAS keywords.

(B) n number of variables

(C) \$ indicates character variable

(D) `variable1...variablen` lists variables (must contain either all numeric or all character)
The ARRAY statement must be used before the arrayname is referenced in the DATA step.

Sample Program 1:

```
DATA RECODE ;
  INPUT A B C D ;
  IF A = 99 THEN A = . ;
  IF B = 99 THEN B = . ;
  IF C = 99 THEN C = . ;
  IF D = 99 THEN D = . ;
DATALINES;
42 43 26 99
43 14 99 34
42 99 53 25
99 34 33 94
;
PROC PRINT ;
RUN;
```

Sample Output 1:	OBS	A	B	C	D
	1	42	43	26	.
	2	43	14	.	34
	3	42	.	53	25
	4	.	34	33	94

Sample 2:

The SAS program in **Sample 1** can be re-coded to reduce the amount of repetitive coding.

Sample Program 2:

```
DATA ARRAY1 ;  
  INPUT  A B C D ;  
  ARRAY NINES(4) A B C D ;  
    DO COUNT = 1 TO 4 ;  
      IF NINES(COUNT) = 99 THEN NINES(COUNT) = . ;  
    END;  
  DROP COUNT ;  
DATALINES;  
42 43 26 99  
43 14 99 34  
42 99 53 25  
99 34 33 94  
;  
  
PROC PRINT ;  
RUN;
```

Sample Output 2:

OBS	A	B	C	D
1	42	43	26	.
2	43	14	.	34
3	42	.	53	25
4	.	34	33	94

Sample 3:

Note: The number of elements in the array may be unknown. In such a case the asterisk replaces the actual number in the ARRAY statement. The **DIM function** returns the number of elements in the array.

Sample Program 3

```
DATA ARRAY2 ;
  INPUT  A B C D ;
  ARRAY NINES(*) A B C D ;
  DO COUNT = 1 TO DIM(NINES) ;
    IF NINES(COUNT) = 99 THEN NINES(COUNT) = . ;
  END;
  DROP COUNT ;
DATALINES;
42 43 26 99
43 14 99 34
42 99 53 25
99 34 33 94
;

PROC PRINT ;
RUN;
```

Sample Output 3 :

OBS	A	B	C	D
1	42	43	26	.
2	43	14	.	34
3	42	.	53	25
4	.	34	33	94

Sample 4:

Use an ARRAY statement to create new variables.

Sample Program 4:

```
DATA ARRAY3 ;  
  INPUT LETTERS $ ;  
  ARRAY NEWVAR(3) $ SUBT1-SUBT3 $ ;  
  DO I = 1 TO 3 ;  
    NEWVAR(I) = SUBSTR(LETTERS, I, 1) ;  
  END ;  
  DROP I ;  
DATALINES;  
ABC  
LPY  
;  
  
PROC PRINT ;  
RUN;
```

Sample Output 4:

OBS	LETTERS	SUBT1	SUBT2	SUBT3
1	ABC	A	B	C
2	LPY	L	P	Y

Sample 5:

Use an ARRAY to accomplish the following tasks:

- (1) If the value of a variable is less than 0, then set it to missing.
- (2) Keep count of the number of times this occurs for each observation.

Sample Program 5:

```
DATA ARRAY4;  
  INPUT X1 - X4;  
  ARRAY TEST(4) X1 - X4;  
  COUNTER = 0;  
  DO NUM = 1 TO 4;  
    IF TEST(NUM) < 0 THEN DO;  
      TEST(NUM) = .;  
      COUNTER + 1;  
    END;  
  END;  
  DROP NUM;  
  DATALINES;  
0 1 3 -6  
1 -7 0 0  
-2 0 -1 5  
1 2 -3 4  
;  
  
PROC PRINT;  
RUN;
```

Sample Output 5:

OBS	X1	X2	X3	X4	COUNTER
1	0	1	3	.	1
2	1	.	0	0	1
3	.	0	.	5	2
4	1	2	.	4	1

Sample 6:

(1) Suppose we have the data lines :

P1	P2	P3	P4	P5
2	0	1	9	1
9	1	2	0	2
0	9	2	0	1

(2) Create a SAS data set and assign the following scores

39.3, 84.4, 77.4, and 47.0

for each of the values 0, 1, 2 and 9 respectively.

(3) Use the SUM function to determine the sum of the scores.

Sample Program 6 :

```
DATA TEST ;
  INPUT P1 P2 P3 P4 P5 ;
  ARRAY REN(5) P1 - P5 ;
  DO J= 1 TO 5 ;
    IF REN(J) = 0 THEN REN(J) = 39.3 ;
    ELSE IF REN(J) = 1 THEN REN(J) = 84.4 ;
    ELSE IF REN(J) = 2 THEN REN(J) = 77.4 ;
    ELSE IF REN(J) = 9 THEN REN(J) = 47.0 ;
  END;
  SCORE = SUM(P1,P2,P3,P4,P5);
  DROP J ;
  DATALINES;
2 0 1 9 1
9 1 2 0 2
0 9 2 0 1
;

PROC PRINT ;
RUN;
```

Sample Output 6:

OBS	P1	P2	P3	P4	P5	SCORE
1	77.4	39.3	84.4	47.0	84.4	332.5
2	47.0	84.4	77.4	39.3	77.4	325.5
3	39.3	47.0	77.4	39.3	84.4	287.4

Four variables used in the SET and MERGE statements

(1) **IN** = variable

Purpose: To determine if the data set contributed to the observation currently in the program data vector. Internally, the special **IN** = variable is set to 1 when data set contributes to the current observation.

(2) **END** = variable

Purpose: To determine if the data set contributed the last observation.
(set to 1 after the last observation is read)

(3) **FIRST.by-variable** **LAST.by-variable**

Purpose: **FIRST.by-variable** and **LAST.by-variable** are automatically created and named by SAS for each variable in the BY statement.

FIRST.by-variable =1 for the first observation in a by-group, otherwise **FIRST.by-variable** =0

LAST.by-variable =1 for the last observation in a by-group, otherwise **LAST.by-variable** =0

Note: The **FIRST.by-variable** and the **LAST.by-variable** appear in the program data vector, but not in the SAS data set.

MERGE with IN = variable and END = variable

General Form: MERGE sasdsname1 sasdsname2 (IN = **variable1**) END = **variable2**; BY
variables ;

|
(A)

|
(B)

(A) **variable1** a temporary numeric variable

The value of **variable1** is 1 when the data set contributes
to the current observation; otherwise, the value of **variable1** is 0.

(B) **variable2** a temporary numeric variable

The value of **variable2** is 1 after the last observation is read.

Sample:

```
DATA NEW2 ;  
  MERGE NAMES CREDIT END = E ;  
  BY NAME ;  
  FIRSTNAM = FIRST.NAME ;  
  LASTNAM  = LAST.NAME  ;  
  END = E ;
```

```
PROC PRINT ;  
  TITLE ' SAS Data Set NEW2 ' ;  
RUN;
```

SAS data set NAMES

OBS	NAME
1	JENNIFER
2	MANUEL
3	TONY

SAS data set CREDIT

OBS	NAME	DATE	CREDIT
1	MANUEL	11780	355
2	MANUEL	11781	125
3	TONY	11842	350

Sample Output:

SAS data set NEW2

OBS	NAME	DATE	CREDIT	FIRSTNAM	LASTNAM	END
1	JENNIFER	.	.	1	1	0
2	MANUEL	11780	355	1	0	0
3	MANUEL	11781	125	0	1	0
4	TONY	11842	350	1	1	1

Sample : The IN = variable

```
DATA MERGE;  
  MERGE NAMES (IN=A) SURVY (IN=B) ;  
  BY NAME ;  
  IF A = 1 AND B = 1 ;
```

```
PROC PRINT;  
RUN;
```

SAS data set NAMES

OBS	NAME	SEX
1	JENNIFER	F
2	MANUEL	M
3	PAUL	M
4	RENEE	F
5	TONY	M

SAS data set SURVY

OBS	AGE	HEIGHT	NAME
1	28	64	JENNIFER
2	35	60	MANUEL
3	35	54	RENEE
4	32	68	TONY

Sample Output :

SAS data set MERGE

OBS	NAME	SEX	AGE	HEIGHT
1	JENNIFER	F	28	64
2	MANUEL	M	35	60
3	RENEE	F	35	54
4	TONY	M	32	68

Sample :

```
DATA ACCTCPU ;  
    SET TEMP8 ;  
    IF FIRST.CPUNUM = 0 AND LAST.CPUNUM = 1 THEN DELETE ;  
    BY ACCT CPUNUM ;  
  
PROC PRINT ;  
RUN;
```

Sample Output :

SAS data set ACCTCPU

OBS	ACCT	CPUNUM
1	MXM1	1
2	MXM1	2
3	NUA1	2
4	NUA1	6
5	NVJ2	1
6	NVJ2	2
7	NVJ2	3
8	NVJ2	6
9	NVJ5	1

SET statement, BY statement with NOTSORTED option

Purpose: The **NOTSORTED** option in the **BY** statement is useful when data are grouped according to the values of a variable (in BY groups), but the groups are not in ascending or descending order.

General Form: BY by-variable **NOTSORTED** ;

|
(A)

(A) **NOTSORTED** indicates that observations with the same BY value are grouped together, but are not sorted

Sample:

The data set WEATHER has the observations with the same month value grouped together (the values for month are in calendar month rather than alphabetical).

SAS data set WEATHER

OBS	MONTH	TMP
1	NOV	72
2	NOV	68
3	DEC	58
4	DEC	60
5	JAN	45
6	JAN	56

```
DATA SUBSET6 ;  
  SET WEATHER ;  
  BY MONTH NOTSORTED ;  
  IF FIRST.MONTH ;
```

```
PROC PRINT ;  
  TITLE 'SAS DATA SET SUBSET6 ' ;  
RUN ;
```

Sample Output:

SAS DATA SET SUBSET6

OBS	MONTH	TMP
1	NOV	72
2	DEC	58
3	JAN	45

PROC FORMAT procedure

SAS user-defined formats

Purpose: Create user-defined **formats**

General Form: PROC FORMAT ;

```
VALUE format-name /*general form of the VALUE statement*/  
    range1 = 'label-1'  
    range2 = 'label-2'  
    .  
    .  
    .  
;
```

format-name assigns a name to the **format**

- cannot be longer than 8 characters
- cannot end with a number
- cannot contain any special characters except the underscore
- cannot be the name of a SAS format
- must start with a \$ if it applies to a character variable

range specifies one or more values, a range of values, or a list of ranges that a variable format can have.

character values must be enclosed in quotes

formatted-value the label can be as long as 200 characters

SAS user-defined informats

Purpose: Create user-defined **informats**

General Form: PROC FORMAT ;

```
    INVALUE informat-name /*INVALUE statement*/  
        range1 = 'informatted-value1'  
        range2 = 'informatted-value2'  
        .....  
    ;
```

informat-name assigns a name to the **informat...**

IMPORTANT...

- must be a valid SAS name
- **cannot be longer than 7 characters**
- must not end with a number
- cannot contain any special characters except the underscore
- cannot be the name of an existing informat
- must start with a \$ if it's character informat data

range **specifies** one or more values, a range of values, or a list of ranges that a variable informat can have.

- character values must be enclosed in quotes

For instance...

range	meaning	example
value	a single value	12
value,...,value	a list of values	12, 24, 68
value-value	a range of values	12-68
range,...,range	a list of ranges	12-24,34-68

Examples:

Assigning number to a character string

- **single numbers**

```
PROC FORMAT ;  
    VALUE QFMT 1 = 'APPROVE'  
              2 = 'DISAPPROVE'  
;
```

- **ranges of numbers**

```
PROC FORMAT ;  
    VALUE AGEFMT  
          LOW - < 0 = 'MISCODED'  
          0 - 12   = 'CHILD'  
          13 - 19  = 'TEEN'  
          20 - HIGH = 'ADULT'  
;
```

- **several values that are not in a range**

```
PROC FORMAT ;  
    VALUE SECFMT  
          1 = 'FEMALE'  
          2 = 'MALE'  
          0, 3 - 9 = 'MISCODED'  
;
```

Assigning character string to another character string

- **character values and ranges of character value**

```
PROC FORMAT ;  
    VALUE $GRADE  
          'A'      = 'GOOD'  
          'B' - 'D' = 'FAIR'  
          'E'      = 'POOR'  
          'I', 'U' = 'SEE INSTRUCTOR'  
          'OTHER'  = 'MISCODED'  
;
```

Create permanent formats

```
Libname libref 'drive:\pathname' ;
```

```
PROC FORMAT LIBRARY = libref ;  
           /* the special reserve word 'LIBRARY' is required */
```

```
    VALUE format-name  
        range = 'formatted value1'  
        range = 'formatted value2'  
        ...
```

```
    ;
```

```
    INVALUE informat-name  
        range1 = 'informattd value1'  
        range2 = 'informattd value2'  
        ...
```

```
    ;
```

Example SAS program...(programmed using SAS on the Mac)

Creates permanent user-defined informat as well as user-defined formats.

```
LIBNAME LIBRARY "MACINTOSH:SASPERMS" ;

PROC FORMAT LIBRARY = LIBRARY ;
  INVALUE abc
    'A' - < 'M' = 1
/* the second value of the first range is excluded... noninclusive
notation*/
    'M' - 'Z' = 2
    OTHER = 3
  ;
  VALUE SMOKEFMT
    1 = 'YES'
    2 = 'NO'
  ;
  VALUE $SEXFMT
    'F' = 'FEMALE'
    'M' = 'MALE'
  ;

DATA LIBRARY.TEMP10 ;
  INPUT INITIAL abc. ID $ SEX $ SMOKER AGE PULSE_1 PULSE_2 ;
  FORMAT SEX $SEXFMT. SMOKER SMOKEFMT. ;
DATALINES;
.      1  M   1   31   62   126
B      2  F   1   20   78   154
L      3  M   2   28   64   128
N      4  F   2   29   96   155
P      5  M   1   21   66   128
W      1  F   1   27   96   265
.      2  M   2   21   68   120
C      3  F   2   42   72   138
;

/*you may also access the user-defined informats and formats in the same SAS program*/
PROC PRINT DATA=LIBRARY.TEMP10;
RUN;
```

Example SAS Log:

```
3  LIBNAME LIBRARY "MACINTOSH:SASPERMS" ;
NOTE: Libref LIBRARY was successfully assigned as follows:
      Engine:          V612
      Physical Name:   MACINTOSH:SASPERMS
4
5  PROC FORMAT LIBRARY = LIBRARY ;
6      INVALUE abc
7          'A' - < 'M' = 1 /* the second value of the first range
8          'M' - 'Z' = 2  is excluded... noninclusive notation*/
9          OTHER    = 3
10     ;
NOTE: Informat ABC has been output.
11     VALUE SMOKEFMT
12         1 = 'YES'
13         2 = 'NO'
14     ;
NOTE: Format SMOKEFMT has been written to LIBRARY.FORMATS.
15     VALUE $SEXFMT
16         'F' = 'FEMALE'
17         'M' = 'MALE'
18     ;
NOTE: Format $SEXFMT has been written to LIBRARY.FORMATS.
19

NOTE: PROCEDURE FORMAT elapsed time was 1.40 seconds

20  DATA LIBRARY.TEMP10 ;
21      INPUT  INITIAL abc. ID $ SEX $ SMOKER AGE PULSE_1 PULSE_2 ;
22      FORMAT SEX $SEXFMT. SMOKER SMOKEFMT. ;
23      DATALINES;

NOTE: The data set LIBRARY.TEMP10 has 8 observations and 7 variables.
NOTE: DATA statement elapsed time was 2.09 seconds

32  ;
33
34  PROC PRINT DATA=LIBRARY.TEMP10;
35  RUN;

NOTE: PROCEDURE PRINT elapsed time was 0.68 seconds
```

Example SAS Output:

The SAS System

OBS	INITIAL	ID	SEX	SMOKER	AGE	PULSE_1	PULSE_2
1	3	1	MALE	YES	31	62	126
2	1	2	FEMALE	YES	20	78	154
3	1	3	MALE	NO	28	64	128
4	3	4	FEMALE	NO	29	96	155
5	3	5	MALE	YES	21	66	128
6	3	1	FEMALE	YES	27	96	265
7	3	2	MALE	NO	21	68	120
8	1	3	FEMALE	NO	42	72	138

